(4)

COMPUTER SYSTEMS LABORATORY

STANFORD UNIVERSITY · STANFORD, CA 94305-4055

Microsupercomputers: Design and Implementation

Stanford University
Computer Systems Laboratory

Technical Progress Report

April 1988 - October 1988

Principal Investigator
John L. Hennessy

Associate Investigator
Mark A. Horowitz

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ADA204969

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) MICROSUPERCOMPUTERS: DESIGN AND IMPLEMENTATION	5. TYPE OF REPORT & PERIOD COVERED Semiannual Technical Progress Report April 1988 - October 1988
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John L. Hennessy and Mark A. Horowitz	8. CONTRACT OR GRANT NUMBER(s) N00014-87-K0828
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford University Department of Electrical Engineering Stanford, CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS R&T Project Code: 4331685
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209-2308	12. REPORT DATE November 1988 / 13. NO. OF PAGES 18
	15. SECURITY CLASS. (of this report) Unclassified
14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office) Office of Naval Research Computer Science Division, Code: 1133 800 N. Quincy St., Arlington, VA 22217-5000	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE None

16. DISTRIBUTION STATEMENT (of this report)

Approved for public release. Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

Publications noted in bibliography attached.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Architecture, Parallel Programming, CAD Tools.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Summary of technical progress.

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Technical Progress Report

April 1988 - October 1988

Contract No. N00014-87-K-0828

Order No. 1133

R & T Project Code: 4331685

Principal Investigator: John Hennessy

Monitored by M. Pullen, J. Toole

This work is supported by the Defense Advanced Research Projects Agency and Office of Naval Research.

Technical Progress

This report summarizes our progress for the period April 1988 - October 1988. This progress is documented in detail in over thirty publications listed in the end.

1 Parallel Processor Architecture

1.1 Basic Architecture Studies

In the past six months, we have been involved in several research efforts dealing with parallel processing applications and architectures [Agarwal 88a, Gupta 88a, Gupta 88b, Gupta 89a, Gupta 89b, Gupta 89c, Gupta 89d, Gupta 89e, Gupta 89f]. In the parallel processing architectures area, our main focus has been on obtaining detailed multiprocessor memory reference traces from several *real* parallel applications, and using them to evaluate implications for parallel architectures. Our paper [Agarwal 88a] uses traces obtained from a VAX-8350 (using modified microcode) to characterize the amount of sharing present in user programs and in the operating system. We also provide data about the temporal, spatial, and processor locality of shared blocks. A major limitation of the microcode-based scheme is that it is not possible to get traces for more than 4 processors, as we are limited by the maximum number of processors on the VAX-8350.

To get over the limitation of 4 processor traces, we developed a new tracing scheme based on the VAX T-bit. Basically, the tracing works as follows. We spawn as many processes as the application desires under the control of a master process. The master process then single steps the application processes in a round robin manner using the VAX T-bit, and while doing that it also records all references made by them. With each reference, the ID of the processor producing it and its type (read/write/ifetch) are recorded. We have now obtained reference traces for several applications, including LocusRoute which is a high quality global router for standard cells, P-Thor which is parallel logic simulation program using the Chandy-Misra algorithm, and MP3D which is a parallel program that determines how shock waves are produced in the upper atmosphere by high-speed objects. We have several 8 processor, 16 processor, and 32 processor traces from these applications.

Other work has focused on tracing and analyzing the synchronization behavior of programs [Davis 88]. As one might expect, synchronization can become a major bottleneck in a large-scale multiprocessor. Surprisingly, we found that load imbalance was the major source of speed-up degradation in programs parallelized at the loop level.

1.2 Scaleable Shared Memory Multiprocessors

The goal of this area of research is to explore architectural approaches that will allow the creation of a large-scale shared memory machine using the fastest available microoprocessors. Our focus has been on directory-based structures, and we have compared their performance to other approaches using traces collected as described in the previous section [Agarwal 88b]. The following table discusses the advantages of this approach compared to bus-based and non-cache-coherent approaches.

Scalability of Shared Memory Architectures

Approach	Do not Cache Shared Data	Cache Shared data. Coherency by directory method	Cache coherency with distributed protocol.
Example Machines	RP3 Ultracomputer	Our Approach	Encore Sequent
Performance Limitation	Limited by potentially shared reference frequency	Limited by (shared reference frequency) X (actual degree of sharing)	Limited by broadcast nature of protocol

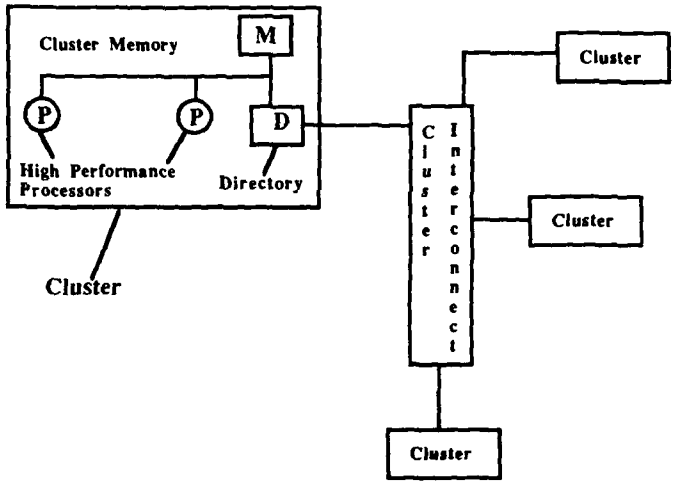
In our paper [Gupta 89e], we use the larger traces to explore the cache invalidation patterns and their impact on a directory-based cache coherence protocols. We propose a classification scheme for data objects found in parallel applications and link the invalidation patterns observed in the traces to these high-level objects. This enables us to get insight into how the invalidation patterns would look when we scale the number of processors. Our results indicate that it should be possible to scale "well-written" programs to a large number of processors without an explosion in invalidation traffic. This study is highly relevant to the multiprocessor architecture that we are building at Stanford. We have also been using the traces for other studies.

In another paper [Gupta 89f], we explore the value of multiple hardware contexts in multiprocessor architectures. We propose that there be a small fixed number (2-4) of hardware contexts per processor. Using a very simple context switch criterion, which is switching contexts on a cache miss or on a write-hit to read-shared data, we show that it is possible to achieve significant performance benefits for some applications.

1.3 Future Work

We are currently in the process of exploring a detailed design using the directory approach. Our machine, illustrated below, would use clusters of processors connected with a switch or a point-to-point network (we are still studying this issue). Each cluster is a bus-based, shared-memory multiprocessor. We are investigating the design of a directory scheme using the Silicon Graphics multiprocessor as a prototype cluster.

Directory-Based Multiprocessor Architecture



1.4 Shared Memory and Message Passing Architectures

On the boundary of architecture and *applications* area, we have been doing research on evaluating the tradeoffs between shared-memory and message-passing parallel architectures. In our paper [Gupta 89e], using the LocusRoute application as a case study, we evaluate the network traffic needed to obtain similar quality of final results in the two architectures. By explicitly varying the frequency of interprocessor updates, the level of traffic in the message passing approach can be reduced by two orders of magnitude, to as little as 1% of the traffic in the shared memory approach while still obtaining solution quality within 10% of the quality given by the shared memory version. We show that exploiting data locality can further lower this message traffic by as much as 67%. To obtain the above traffic figures the message passing version, however, requires significantly larger programming effort. We are still exploring how the above results might generalize to other applications.

2 Parallel Software

In the applications area, we have again been working on several fronts. We have started a new project that explores parallel implementations of assumption-based truth-maintenance systems (ATMS), a highly popular and compute intensive paradigm in Artificial Intelligence. We began with doing a C-based implementation of the ATMS, which is now complete and runs significantly faster than the currently available lisp-based implementations. We also have a parallel version now running on the Encore Multimax. Our results, reported in [Gupta 89d], show only limited speed-up so far, and we are looking into the possible bottlenecks. While on the subject of AI applications, we are continuing our collaborative efforts with CMU on the parallel implementations of OPS5 rule-based systems. A paper detailing our implementation and speed-ups [Gupta 88b] is to appear in the Journal of Parallel Programming. At AAAI-88 [Gupta 88a], we also presented some new results detailing how to map rule-based systems on to message-passing architectures. The key data structure in our mapping is a distributed concurrent hash table which distributes both data and work between processors.

Another application area that we have been exploring is VLSI CAD. We are exploring the use of the Chandy-Misra distributed simulation algorithm as applied to the domain of logic simulation. We have now gathered up several realistic benchmark circuits, and have a parallel implementation running. In [Gupta 89c], we present data characterizing the intrinsic parallelism in the benchmark circuits using the generic Chandy-Misra algorithm. Our results show that the average number of logic elements available for concurrent execution ranges from 6.2 to 92 for the benchmark circuits, with an overall average of 50. Although this is twice as much parallelism as that obtained by traditional event-driven algorithms, we feel it is still too low. One major factor limiting concurrency is the large number of global synchronization points --- "deadlocks" in the Chandy-Misra terminology --- that occur during execution. Towards the goal of reducing the number of deadlocks, we present a classification of the types of deadlocks that occur during digital logic simulation and some domain specific methods for reducing them. For one of the benchmark circuits, the use of the proposed techniques eliminates all deadlocks and increases the average parallelism from 40 to 160.

In [Gupta 89a], with David Black from CMU, we explore new algorithms for dynamically replicating and migrating pages in a distributed memory architecture like the BBN Butterfly. Simulations show that our algorithms, which require some hardware support, can result in several fold speed-up over random page assignment.

3 Uniprocessor Architecture

3.1 Super Scalars

In addition to our work on multiprocessors, we have also been investigating how much parallelism is available at the lowest level -- in the base instruction stream of a processor. Unlike most of the previous work in this area, we have focused on non-scientific applications, i.e., compilers and CAD programs that we run on our workstations. These programs are known for their frequent branches and complex control flow. Our initial work in this area [Smith 89] showed that there is about a factor of 2 parallelism at this level, but one of the big problems in getting this speedup was in fetching instructions.

To achieve a CPI under 1, the machine must fetch more than one instruction per cycle. Branches cause two main problems for the fetcher. The first is the delay between fetching the branch and determining the destination. This problem exists for current machines, but gets worse as the CPI goes down. The second problem is due to alignment of the basic blocks. Since the branches are not aligned on double (or quad) word boundaries, when the machine fetches a group of code not all the instruction fetched will be executed. Together these problems mean a machine with a simple fetcher of 2 instructions can have a maximum instruction rate of 1.4. We are working on two methods to reduce these fetch limits, one relies on hardware and one that relies on software.

The hardware approach uses a 4 wide fetcher to reduce the alignment problems and an integrated branch target buffer/instruction cache to reduce the delay associated with the branch. Together this hardware has a fetch efficiency of over two instructions/cycle, and greatly reduces the fetch effects on the machine performance. The software approach assumes that the processor always fetches two instructions aligned on double-word boundaries, and tries to align branches and fill the slots after a branch (they can be as many as 3 instructions that need to be placed after the branch). Using this technique we have gotten good fetch efficiencies of over 1.7 instructions/cycle with a moderate cost in code expansion. We are continuing to refine both instruction fetch models as well as looking at how to build the actual execution units.

3.2 High-Performance Cache Design

It has become clear that higher performance uniprocessors and multiprocessors will have increased dependence on cache performance. In our multiprocessor, cache miss traffic is still a major component of the shared memory traffic for *realistic* applications. Thus, we are continuing to examine cache organizations that further reduce this traffic, since it severely limits a multiprocessor performance. Recent work has included the first cache study completely based on system performance as the comparison metric [Przybylski 88a, Przybylski 88b].

In addition, we have begun a study of using software techniques to optimize cache performance. This work is focusing on both instruction and data caches. We have already demonstrated significant improvement for instruction caches [McFarling 89]. The potential for data cache optimization on large scientific programs is even larger; preliminary experiments have shown greater than a factor of two in running time--a factor that could be easily multiplied on a multiprocessor.

4 Computer-Aided Design (CAD) Tools

This effort focuses on new design tools to decrease design time and on the use of parallelism in tools.

4.1 Computer-Aided Synthesis

Within the last six months progress has been made on the HERCULES program, originally presented at the ARPA meeting in Snowbird. HERCULES produces logic-level circuit descriptions from a high-level hardware description language, called HardwareC. An overview of HERCULES is in a paper published in the DAC proceedings of 1988 [De Micheli 88]. Hercules was also demonstrated at the DAC university booth. A language reference for HardwareC is also enclosed, and was published as a Stanford technical report [Ku 88].

Hercules has been used to design an interface chip, called DAIO, between a 32-bit workstation and a compact disk player, following the AES protocol. A design of a DCT chip and an encoder chip for the space telescope project at Stanford (MAMA) are also being designed using HERCULES.

Another project on register-transfer logic synthesis has scored some good initial results, which are still unpublished. The goal of this research is to devise logic synthesis techniques that do not partition combinational logic from register, i.e. that effect logic optimization based on logic transformation on a register transfer level descriptions. The new algorithms have been recently implemented in the program MINERVA.

4.2 Incremental Simulation

Simulation now requires vast amounts of cpu time. This severly limits the size of a design that can be tested thoroughly. Incremental simulation is a possible solution to these limit. Specific synthesis and analysis tools are also proposed.

We proposed two incremental simulation algorithms, the *incremental-in-space* and *incremental-in-time* algorithms, and were implemented in our *THOR* simulation system [Hwang 88]. These two algorithms are comparable to each other: one shows better performance for some circuits over the other, depending on the circuit structure and topology of the circuit under simulation.

4.3 Behavioral Synthesis: Hermod System

The primary purpose of this project is to allow the designer to make changes and optimizations at as high a level as possible, while allowing him to observe the ramifications of his changes at a lower level. The designer can also provide help to the synthesis routines in the selection of a good design [Odani 89]. This user guidance is necessary because of the huge design search space faced by high-level synthesis programs. The synthesis system displays the data/control flow graph extracted from a functional model on the window screen. The register-transfer representation of a behavioral level description is displayed on the screen after optimization by the system.

Partitioning in behavioral synthesis will be investigated next. In VLSI design, it is important to

partition the hardware at the early stage of design to generate good quality designs. Partitioning of algorithmic/behavioral descriptions should provide the synthesizer with the capability to explore design space effectively. Algorithmic partitioning can be achieved by splitting a procedure into multiple processes that can be executed concurrently or be pipelined. An algorithmic partitioner will be designed for behavioral descriptions written HardwareC, the high level language for Hercules synthesis system.

4.4 Automatic Layout

Using parallel processing, the quality of automatic layout can be greatly enhanced. The Locus project works to this end, and the Simulated Annealing research relates understanding the important optimization strategies.

4.4.1 Parallel Place and Route: The Locus Project

The parallel router is complete. Recent progress on the router has improved the speedups to a range of 10 to 15 using 15 processors, using one axis of parallelism. When combined with a second axis of parallelism, we expect speedups of up to 61 times using 120 processors. The combined multiplicative effect used to predict this performance has been demonstrated on the 15 processor Encore MULTIMAX, and improved further with the use of dynamic scheduling of tasks [Rose 89a]. Earlier versions of this work have been published. The first focuses on the CAD algorithm [Rose 88a] while the second is about the parallel aspects [Rose 88b].

Work has begun on the Locus placement program. We have developed a new placement algorithm that incorporates the parallel global router. It combines a number of the best features of two of the most popular placement algorithms - Min-Cut placement and Simulated Annealing, and has the added advantage of a cost function that is closer to the true final objective - area.

4.4.2 Simulated Annealing

One way to alleviate the high computational cost of Simulated Annealing is to replace part of it with a faster heuristic, and then follow it with lower-temperature Simulated Annealing. A crucial parameter in this kind of approach is the temperature at which to begin the Simulated Annealing phase. This work addresses that problem, in the context of the automatic placement problem, by developing a method of measuring the temperature of a given placement. It appears, in short form, in ICCAD 89 [Rose 88c], and has been submitted in its entirety in [Rose 89b].

4.5 General Compiled Electrical Simulation

Weise has been investigating advanced compiler techniques for speeding up electrical simulation such as the type SPICE performs [Weise 89]. The major idea behind this work is that by compiling together a simulator and a circuit one can produce a *simulation program* that, when run, yields the same result as applying the simulator to the circuit, but which runs many times faster than the simulator.

The initial results of this research are very encouraging. The system can already perform a transient analysis of linear circuits. Experiments have been conducted on circuits containing up to 120 components. The simulation programs produced by the system run about 13 times faster

than Spice3. Extrapolating from this result and from other factors, Weise believes a system with the same functionality as Spice3 but at least 5 times the speed can be built.

More importantly, the simulation programs produced by the system can be parallelized much more effectively than a standard Spice-like simulator can be parallelized. Future research includes compiling simulation programs for execution on parallel machines and the design of special purpose hardware for executing simulation programs. Weise anticipates that a board that plugs into a workstation and that sustains 100 MFLOPS on Spice can be built for around $15,000.

5 VLSI

We have been continuing our efforts in exploring ways to use IC technologies to help build high speed computer systems. In this effort we have been exploring both novel circuit structures, and new technologies to improve overall system performance. Our recent effort has been focused in four areas: high speed RAMS, BiCMOS, floating-point, and testing.

5.1 RAM Design

We are continuing to work on very high speed RAM design, because we feel that RAM speed is the major limitation on building very fast uniprocessors. For next generation ECL systems, RAM speed must be under 4ns which is about 2x smaller than current generation parts. In addition, recent work on cache design [Przybylski 88a] has shown that small, fast caches are not a good design point -- a machine with a larger, but slightly slower cache (and hence cycle time) can actually have higher overall performance. Thus, we are trying to design a reasonably large (64K bit) fast (under 4ns) BiCMOS sRAM. We plan to use the CMOS Storage, Emitter Access (CSEA) memory cell that was mentioned in the last progress report [Yang 88a, Yang 88b]. This cell couples a CMOS latch with an emitter follower to form a memory cell with no dc power, but a very fast low voltage swing access path. The initial prototype was a 4K bit RAM fabricated in a 1.5μ BiCMOS technology by IDT. We are now designing a 64K bit RAM, in a .8μ BiCMOS technology. TI has agreed to fabricate the part when the design is completed, which we expect to be during the summer of 89.

The new RAM is being tailored to be used as a cache RAM in a high speed computer system. To improve the performance of both the RAM and the system, we customize the interface of the RAM. For example the RAM will be clocked, and the timing of writes in to the RAM will be generated internally. In addition, the RAM will probably provide a double width write port to allow the secondary cache to do double wide refills at half the processor frequency. This makes both the RAM design easier (high speed writes are hard) and should also simplify the design of the secondary cache (eliminating the need for a very high freq. multiplexor.) To get feedback on the RAM architecture we have been talking with designers at both SUN and MIPS Computer Systems.

5.2 BiCMOS

In addition to our work on the BiCMOS RAM, we are exploring ways of building other high performance circuits in BiCMOS. In cooperation with Signetics Corporation, and Texas Instruments, we have designed and fabricated a number of adder structures, and have come up with promising designs for PLAs and various bus drivers. One of our first efforts was on a new BiCMOS Manchester carry circuit [Rosseel 89]. It uses a bipolar sense transistor that is kept out of saturation by using a current mirror arrangement. The performance of the circuit is quite good, with subnanosecond delays for 4 bit lookahead. We have also looked at how BiCMOS changes the architecture that one might use for an adder. For example, since BiCMOS can drive large cap loads, a Brent Kung tree adder can be modified to remove the need for the fanout tree. This reduces the number of stages in the adder by almost a factor of two and greatly speeds up the device. We have built a prototype adder in TI .8μ BiCMOS technology, and have gotten 32 bit add times of around 6ns. We think our measurement system is actually limiting the device performance, and are currently working on building a better test setup.

To support our design effort we are working on a set of CAD tools to support BiCMOS. As mentioned in the last report, we have modified Magic to handle BiCMOS designs and are working on various analysis tools for BiCMOS. Our work on BiCMOS simulation is continuing, and we have a bipolar switch-level simulator, Bisim [Kao 88] up and running. We are now working on incorporating MOS devices into its model, and hope to have a workable simulator by the end of the school year.

5.3 Multiplication

The iterating array multiplier that we described in the last report is a fast, area-efficient method of multiplication. We have continued our work on this structure, first looking at methods to improve the performance of the basic array, and second by looking at the issues involved in making an IEEE compatible multiplier using this architecture.

To improve the performance of the base tree, we are creating a better internal clock generator and using different full-adder and latch structures. These improvements should increase the iteration frequency to over 100MHz in a 1.6μ CMOS technology. Our results also show that replacing the Booth Encoders and Muxes with an extra level in the tree is a good idea. Because the Booth Muxes are so large, the change is about area neutral, but it greatly reduces the number of cell types used in the multiplier. It also reduces the number of clocks needed to complete a multiplier [Santoro 89].

The main issue with meeting the IEEE floating-point standard is rounding. To generate a correctly rounded result, the multiplier must accumulate all the lower order bits in the partial products to generate a correct carry in to the final carry propagate adder. In an iterating scheme, this carry can only be generated late -- one or two clocks after the carry-save output is available. We have developed a scheme which allows the CP add to be done before the carry and overflow bits are known, and the efforts of these bits can be used late in the operation to select the correct result. This method requires only slightly more hardware than that needed when the carry input is known early, and will allow fast IEEE multiplies to be done in iterating structures.

5.4 Single-Chip Testers

We have continued working on building a single chip tester. Our work on integrated pin electronics [Gasbarro 88, Gasbarro 89] was very successful; we were able to generate subnanosecond timing in a 2μ CMOS technology. The next stage of the project was the design of a complete tester on a chip. This chip, called Testarossa, contains a dRAM for vector storage, a decompressor to increase the effective vector size, and the pin electronics for 16 DUT pins. The dRAM stores 40K bits which is 2K raw vectors per pin. With a decompression rate of 3-5, this chip should be able to store between 6-10K vectors per pin. It uses a per-pin architecture, allowing each pin to be individually configured. The chip provides a choice of two output high levels and two output low levels for each DUT pin, with an edge placement accuracy of about 1ns.

The chip was designed using a set of design tools developed at Xerox, and was fabricated over the summer. The chips are fully functional, although there seems to be a small error in the dRAM interface that we are currently debugging. The chip is about 11mm on a side and contains about 200K transistors. We are in the process of cleaning up the design, and plan to

resubmit it during the beginning of the year. With the improved chips, we will build a high performance low cost tester that will be used at Stanford. Using these chips it should be possible to build an IMS class tester using only 8 chips. In addition, because the tester is so small in size, it will be possible to eliminate the problem with reflections on the cables between the pin-electronics and the DUT pin by removing the cable.

5.5 Programmable Gate Arrays

We have begun to investigate architectures for a new and exciting idea in integrated circuits: the programmable gate array (PGA). It reduces the IC manufacturing time from months to *minutes*, and prototype cost from tens of kilodollars to under $100. A PGA is similar to a gate array in structure, but can be field-programmed to specify the function of the basic logic blocks and their interconnection. The initial work investigates the effect of the complexity of the logic block on the area of the resulting programmable gate array [Rose 89c].

6 Staff

Faculty:

John Hennessy	Principal Investigator	jlh@vsop.stanford.edu
Mark Horowitz	Co-principal Investigator	horowitz@mojave.stanford.edu
Tom Blank	Assistant Professor	blank@mojave.stanford.edu
Giovanni De Micheli	Assistant Professor	nanni@mojave.stanford.edu
David Dill	Assistant Professor	dill@amadeus.stanford.edu
Anoop Gupta	Assistant Professor	ag@amadeus.stanford.edu
Monica Lam	Assistant Professor	lam@mojave.stanford.edu
Daniel Weise	Assistant Professor	daniel@mojave.stanford.edu

Research Associates and Post-doctoral Scholars:

M. Ganapathi	Research Associate
K. Gopinath	Post-doctoral Scholar
S. Hwang	Research Associate
M. Morf	Visiting Scholar
S. Przybylski	Post-doctoral Scholar
J. Rose	Research Associate

Graduate Students:

J. Acken	Electrical Engineering
R. Alverson	Electrical Engineering
J. Basu	Computer Science
M. Blatt	Computer Science
R. Chandra	Computer Science
Y. Cho	Electrical Engineering
H. Davis	Computer Science
J. Gasbarro	Electrical Engineering
K. Gharachorloo	Electrical Engineering
M. Hailpern	Computer Science
B. Hayes	Computer Science
R. Kao	Electrical Engineering
D. Ku	Electrical Engineering
J. Laudon	Electrical Engineering
S. McFarling	Electrical Engineering
M. Martonosi	Computer Science
S. Nowick	Computer Science
S. Richardson	Electrical Engineering
G. Rosseel	Electrical Engineering
E. Rothberg	Computer Science
A. Salz	Electrical Engineering
M. Santoro	Electrical Engineering
L. Sha	Electrical Engineering
R. Simoni	Electrical Engineering

J. Singh	Electrical Engineering
M. Smith	Electrical Engineering
L. Soule	Electrical Engineering
D. Stark	Electrical Engineering
S. Tjiang	Computer Science
A. Tucker	Computer Science
J. Vera	Electrical Engineering
W. Weber	Computer Science
M. Wing	Electrical Engineering
D. Wingard	Electrical Engineering
M. Wolf	Electrical Engineering

References

[Agarwal 88a] Agarwal, A., Gupta, A.
Memory-Reference Characteristics of Multiprocessor Applications under MACH.
In *SIGMETRICS*. IEEE, May, 1988.

[Agarwal 88b] Agarwal, A., Simoni, R., Hennessy, J., Horowitz, M.
Scaleable Directory Schemes for Cache Consistency.
In *15th International Symposium on Computer Architecture*. IEEE, Honolulu, HI, June, 1988.

[Davis 88] Davis, H., Hennessy, J.
Characterizing the Synchronization Behavior of Parallel Programs.
In *Sym. on Parallel Programming: Experience with Applications, Languages and Systems*. ACM , New Haven, CT, July, 1988.

[De Micheli 88] De Micheli, G., Ku, D.
HERCULES - A System for High-Level Synthesis.
In *Design Automation Conference*. IEEE/ACM, Anaheim, CA, June, 1988.

[Gasbarro 88] Gasbarro, J., Horowitz, M.
Integrated Pin Electronics for VLSI Functional Testers.
In *Custom Integrated Circuits Conference*, pages 16.2.1-16.2.4. IEEE, Rochester, NY, May, 1988.

[Gasbarro 89] Gasbarro, J., Horowitz, M.
Integrated Pin Electronics for VLSI Functional Testers.
IEEE Journal of Solid State Circuits , April, 1989.
To be published.

[Gupta 88a] Gupta, A., Tambe, M.
Suitability of Message Passing Computers for Implementing Production Systems.
In *AAAI-88*. St. Paul, MN, August, 1988.

[Gupta 88b] Gupta, A., Forgy, C., Kalp, D., Newell, A., Tambe, M.
Parallel Implementation of OPS5 on the Encore Multiprocessor: Results and Analysis.
Intl. Journal of Parallel Programming , 1988.
To appear.

[Gupta 89a] Black, D., Gupta, A., Weber, W-D.
Competitive Management of Distributed Shared Memory.
In *Compcon*. ACM, March, 1989.
To appear.

[Gupta 89b] Weber, W-D., Gupta, A.
Analysis of Cache Invalidation Patterns in Multiprocessors.
In *ASPLOS-III*. ACM/IEEE, Boston, MA, April, 1989.
To appear.

[Gupta 89c] Soule, L., Gupta, A.
Characterization of Parallelism and Deadlocks in Distributed Logic Simulation.
1989.
Submitted for publication.

[Gupta 89d] Rothberg, E., Gupta, A.
Experiences Implementing a Parallel ATMS on a Shared-Memory Multiprocessor.
1989.
Submitted for publication.

[Gupta 89e] Martonosi, M., Gupta, A.
Shared-Memory vs. Message-Passing Architectures: An Application Based Case Study.
1989.
Submitted for publication.

[Gupta 89f] Weber, W-D., Gupta, A.
Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture.
1989.
Submitted for publication.

[Hwang 88] Hwang, S.Y., Blank, T., Choi, K.
Fast Functional Simulation: An Incremental Approach.
IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems CAD-7(7):765-774, July, 1988.

[Kao 88] Kao, R., Alverson, R., Horowitz, M., Stark, D.
Bisim: A Simulator for Custom ECL Circuits.
In *Intl. conference on Computer Aided Design*, pages 62-65. IEEE, Santa Clara, CA, November, 1988.

[Ku 88] Ku, D., De Micheli, G.
Hardware C - A Language for Hardware Design.
Technical Report CSL 88-362, Stanford University, August, 1988.

[McFarling 89] McFarling, S.
Program Optimization for Intruction Caches.
In *ASPLOS-III*. ACM/IEEE, Boston, MA, April, 1989.
To appear.

[Odani 89] Odani, M., Hwang, S.Y., Blank, T., Rokicki, T.
The Hermod Behavioral Synthesis System.
Journal of Systems and Software , 1989.
To appear.

[Przybylski 88a] Przybylski, S., Horowitz, M., Hennessy, J.
Performance Effects in Memory Hierarchy Design.
In *15th International Symposium on Computer Architecture*, pages 290-298. IEEE, Honolulu, HI, June, 1988.

[Przybylski 88b] Przybylski, S.
Performance-Directed Memory Hierarchy Design.
PhD thesis, Stanford University, September, 1988.
Also published technical report number CSL-TR-88-366.

[Rose 88a] Rose, J.S.
LocusRoute: A Parallel Global Router for Standard Cells.
In *25th Design Automation Conference*, pages 189-195. IEEE/ACM, Anaheim, CA, June, 1988.

[Rose 88b] Rose, J.S.
The Parallel Decomposition and Implementation of an Integrated Circuit Global Router.
In *Sigplan Symposium on Parallel Programming*, pages 138-145. ACM, New Haven, CT, July, 1988.

[Rose 88c] Rose, J., Klebsch, W., Wolf, J.
Temperature Measurement of Simulated Annealing Placements.
In *Intl. Conference on Computer-Aided Design.* IEEE, November, 1988.
To appear.

[Rose 89a] Rose, J.
Parallel Global Routing for Standard Cells.
IEEE Trans. on Computer-Aided Design of Circuits and Systems , 1989.
Submitted for publication.

[Rose 89b] Rose, J., Klebsch, W., Wolf, J.
Temperature Measurement and Equilibrium Dynamics of Simulated Annealing Placements.
IEEE Trans. on Computer-Aided Design of Circuits and Systems , 1989.
Submitted for publication.

[Rose 89c] Rose, J., Francis, R., Chow, P., Lewis, D.
The Effect of Logic Block Complexity on Area of Programmable Gate Arrays.
In *Custom Integrated Circuits Conference.* IEEE, 1989.
Submitted for publication.

[Rosseel 89] Rosseel, G., Horowitz, M., Cline, R., Dutton, R.
A Single-ended BiCMOS Sense Circuit for Digital Circuits.
In *Intl. Solid State Circuits Conference.* IEEE, February, 1989.
To appear.

[Santoro 89] Santoro, M., Horowitz, M.
SPIM: A Pipelined 64 x 64 Bit Interative Multiplier.
IEEE Journal of Solid State Circuits , 1989.
To be published.

[Smith 89] Smith, M., Johnson, M., Horowitz, M.
Limits on Multiple Instruction Issue.
In *ASPLOS-III.* ACM/IEEE, Boston, MA, April, 1989.
To appear.

[Weise 89] Weise, D., Seligman, S.
General Compiled Electrical Simulation.
1989.
Submitted for publication.

[Yang 88a] Yang, T., Horowitz, M., Wooley, B.
A 4ns 4kxl Two-Port BiCMOS SRAM.
In *Custom Integrated Circuits Conference*, pages 4.7.1-4.7.4. IEEE, Rochester, NY, May, 1988.

[Yang 88b] Yang, T.S., Horowitz, M., Wooley, B.
A 4nsec, 4Kx1bit, Two-Port BiCMOS SRAM.
IEEE Journal of Solid State Circuits , October, 1988.